Comparing Methods to Identify Defect Reports in a Change Management Database

July 2008

Elaine Weyuker
Tom Ostrand
AT&T Labs – Research

Overall Goal of Research

To determine which files of a large industrial software system with multiple releases are particularly likely to contain the most faults.

This obviously requires knowing what is meant by “containing a fault”.

Why is this important?

This should allow us to build highly dependable software systems more economically by allowing us to better allocate testing effort and resources. It lets testers prioritize and also alerts developers that they may need to re-architect a software entity.

Defining a Fault

We have been using the development stage to determine whether software modifications represent faults or merely a change such as the addition of new functionality or specification changes. Since system testers' sole job function is to find bugs, a change made in response to a modification request of a tester is considered a fault.

Using Keywords to Identify Faults

Other fault prediction researchers have used keywords to identify which software changes represented faults, especially when studying open-source software. Some of the keywords used include: bug, fix, patch, defect and variants such as bugs or fixed.

	Devel Stage Only	Keyword Only	Both	Total MRs
Release A	29	16	19	270
Release B	47	28	32	426
Release C	79	2	36	519
Release D	34	1	20	254

Development Stage Only || Keyword Only

	Fault	Non Fault	% Corr Fault	Fault	Non Fault	% Corr Fault
Release A	25	4	80%	7	9	44%
Release B	45	2	96%	0	28	0%
Release C	75	4	95%	0	2	0%
Release D	34	0	100%	0	1	0%

Misclassifications

- Type I (false positive) – those instances in which an MR that is not a fault MR is incorrectly classified as a fault MR.

- Type II (false negative) - those instances in which an MR that really is a fault MR is incorrectly classified as not being a fault MR.

Understanding Misclassifications

- Need both types of misclassification rates to be low or could manipulate to make things look good. (If identify no MRs as faults then no Type I misclassifications; if identify every MR as a fault then no Type II misclassifications.)

Computing Misclassification Rates

$$\frac{\text{(number of misclassifications of a given type)}}{\text{(total number of MRs classified as faults by that definition)}}$$

Misclassifications

	Type I		Type II	
	Develmt Stage	Keyword	Develmt Stage	Keyword
Release A	8.3%	25.7%	3.2%	10.6%
Release B	2.5%	46.7%	0.0%	12.3%
Release C	3.5%	5.3%	0.0%	15.6%
Release D	0.0%	4.8%	0.0%	14.6%

Where Did the Keywords Go Wrong?

- Initializing all new files into the release – used the phrase “patch-build”
- ``Fixed elsewhere'' or that “nothing needed to be fixed”.
- Describing defective data or fixing data.
- “Fix” referring to the manual or documentation – change in specifications or requirements.
- Negative: not a bug, nothing to be fixed, etc.

Conclusions

- Based on this case study, it appears that the use of keywords does not improve the fault identification rate, and that the development stage provides a much more accurate way of identifying what is and is not a fault.

- For open-source software the use of keywords might be the best available alternative way of identifying fault MRs.